TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Informaatika instituut

ITI40LT

Andres Suislepp 134930IAPB

# KOTLINI VÕIMALUSED ANDROIDILE ARENDAMISEKS REAALAJAS MÄRULIMÄNGU NÄITEL

Bakalaureusetöö

Juhendaja: Roger Kerse
Tehnikateaduste magister
Lektor

Tallinn 2016

# Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Andres Suislepp

23.05.16

# Annotatsioon

Käesoleva bakalaureusetöö eesmärkideks on uurida programmeerimiskeele Kotlin eeliseid ja puuduseid Android platvormile arendamisel, uurida populaarseid reaalaja serverilahendusi ning arendada reaalajas toimuv mobiilimäng kahe mängija vahel kasutades selleks Kotlinit ja eelistatud serverilahendust.

Autor uuris Kotlini sobivust Androidi platvormiga, analüüsides selle häid ja halbu omadusi. Autor leidis, et Kotlini turvaline tüübisüsteem võimaldab arendajatel luua töökindlamaid rakendusi väiksema vaevaga. Kotlini programmikood on hästi loetav ja ühildub ilma probleemideta olemasolevate Java teekidega. Autor käsitles ka erinevaid serverilahendusi nagu Firebase, Meteor, Socket.IO ja Feathers.js, mis pakuvad võimalust realiseerida reaalajas mängu mitme mängija vahel.

Töö tulemusena valmis mobiilimäng, mis on kirjutatud programmeerimiskeeles Kotlin ja mis kasutab serverilahendust Firebase. Autor kirjeldab mängu loomiseks kasutatud põhilisi komponente ja toob näiteid nende implementeerimisest loodud mängu programmikoodis. Autori kogemus Kotlini kasutamisel oli üksnes positiivne ja ühtegi piirangut see arendusele ei seadnud.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 39 leheküljel, 5 peatükki, 25 joonist, 1 tabel.

# Abstract

Kotlin opportunities for Android development in the example of realtime action game

The purpose of this thesis is to research and analyze the programming language called Kotlin, it's suitability for Android development, study different popular solutions for implementing realtime game and to create a fully functional mobile game using preffered solution.

The Author analyzed a programming language Kotlin, it's build processes and usage, pointed out it's positive and negative aspects by comparing them with very popular programming language Java. The Author stated that the biggest advantage of using Kotlin is it's safety which raises application's reliability. The Author also brought out that Kotlin's code is very concice, readable and has no issues operating with existing Java code, which allows using Kotlin code alongside with Java, while the only downsides being noticeably slower compile times and poor documentation. The Author also researched the methodologies of building realtime games. Four popular realtime server-side solutions, Firebase, Meteor, Socket.IO and Feathers.js were thoroughly analyzed. Author preffered Firebase based on its convenience and handiness.

As a result of this thesis a realtime mobile game was created, using Kotlin as the only programming language. The game is played realtime between two players, using Firebase as the server-side solution. The Author describes the main used components for creating the game and gives examples of their implementation. The Author did not notice any drawbacks using Kotlin and states that the experience was only positive.

The thesis is in Estonian and contains 39 pages of text, 5 chapters, 25 figures, 1 table.

# Lühendite ja mõistete sõnastik

| | |
|---|---|
| ***Android Manifest*** | Androidi rakenduse juurkataloogis asuv dokument, mis kirjeldab antud rakendust [1] |
| ***API*** | ***Application Programming Interface***<br>Rakendusliides ehk reeglistik juba eksisteeriva programmiga suhtlemiseks [2] |
| ***APK*** | ***Android Application Package***<br>Androidile omane pakkimisviis rakenduste levitamiseks ja installeerimiseks [3] |
| **Avalda-telli muster** | ***Publish-subscribe pattern***<br>Võrkudele orienteeritud sõnumite saatmise arhitektuuriline muster [4] |
| ***Bitmap*** | Rasterpilt |
| ***DDP*** | ***Distributed Data Protocol***<br>Kliendi ja serveri vahelist suhtlust reguleeriv protokoll [5] |
| ***DSL*** | ***Domain specific language***<br>Spetsiifiline programmeerimiskeel, mis pakub lisa funktsionaalsust juba olemasolevale keelele [6] |
| **ES6** | ***ECMAScript 6***<br>Programmeerimiskeel. |
| ***HTTP*** | ***Hypertext Transfer Protocol***<br>Veebiserveri ja kliendi omavahelise suhtluse protokoll |
| ***JSON*** | ***Javascript Object Notation***<br>Formaat, mille abil kirjeldatakse võti-väärtus stiilis andmeid. |
| ***JVM*** | ***Java Virtual Machine***<br>Java Virtuaalmasin, mis võimaldab arvutil jooksutada Java programme |
| **Klient** | ***Client***<br>Klient. Riist- või tarkvara, mis suhtleb teenuste kaudu serveriga [7] |

| | |
|---|---|
| **klient-server** | ***Client-server***<br>Võrgumudel |
| ***NFC*** | ***Near Field Communication***<br>Kontaktivaba side seadmete vahel, tuues nad vähemalt 4 sentimeetri lähedusele üksteisest [8] |
| ***Node.js*** | Avatud lähtekoodiga populaarne süsteem arendamaks serveripoolseid rakendusi [9] |
| ***NoSQL*** | ***NoSQL***<br>Mitte-relatsiooniline andmebaas |
| ***Null*** | Programmeerimises kasutatav mõiste defineerimaks väärtustamata, määratlemata, tühja või tähenduseta väärtust [10] |
| ***P2P*** | ***Peer-to-peer***<br>Võrdsete õiguste ja võimalustega võrgusõlmede kogum [11] |
| **Pilv** | ***Cloud***<br>Andmete hoidmis mudel, kus digitaalseid andmed on kätte saadavad veebist, kuid nende füüsiline hoiustamine on hajutatud mitme erineva serveri vahel [12] |
| ***Plugin*** | ***Plugin***<br>Tarkvara komponent, mis lisab uut funktsionaalsust juba eksisteerivale programmile [13] |
| **REST** | ***Representational state transfer***<br>Tarkvaraarhitektuuri stiil [14] |
| ***SSE*** | ***Server-sent events*** [15]<br>Tehnoloogia, mille puhul klient saab automaatseid uuendusi serverilt kasutades HTTP ühendust. |
| **Teek** | ***Library***<br>Kollektsioon juba implementeeritud funktsionaalsust, mis on mõeldud korduvkasutamiseks erinevates programmides. [16] |
| ***Websockets*** | Protokoll, mis pakub täielikku kahepoolset kommunikeerimis kliendi ja serveri vahel [17] |

# Sisukord

# Jooniste loetelu

# Tabelite loetelu

# 1 Sissejuhatus

Käesoleva bakalaureusetöö eesmärgiks on analüüsida Kotlini keele võimalusi ja sobivust Androidi platvormile arendamiseks, luues täielikult funktsioneeriva mobiilimängu kasutades programmeerimiskeelt Kotlin.

Töö teises peatükis uurib autor Kotlini eripärasusi ning võrdleb neid programmeerimiskeelega Java. Autor uurib ka, kuidas sobib Kotlin Androidile arendamiseks ning kuidas see abistab Androidi platvormile arendamist.

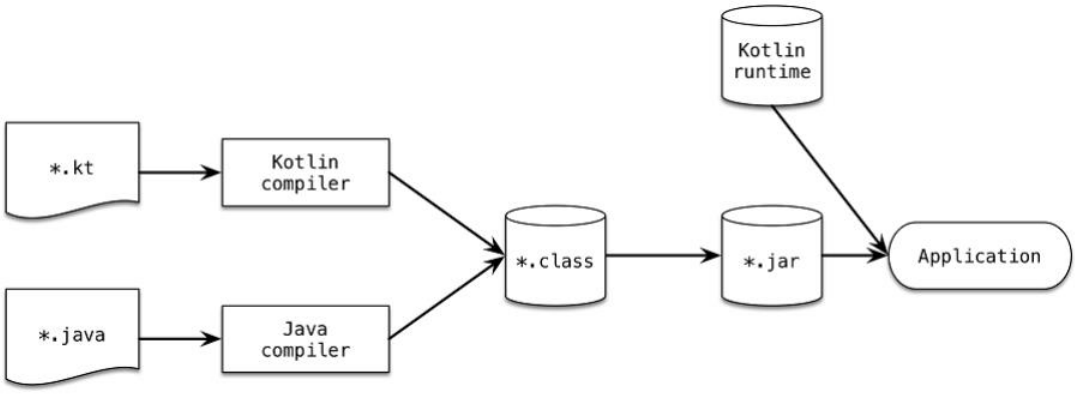
Kolmandas peatükis analüüsib autor tehnoloogiaid, mis võimaldavad luua reaalajas mänge. Autor annab põgusa ülevaate antud metoodikate ideoloogiast. Uuritakse erinevaid serverilahendusi ja- raamistikke, mis võimaldavad luua reaalajas mängitavat mängu. Välja on toodud neli erinevat raamistikku ning analüüsitud nende positiivseid ja negatiivseid omadusi.

Töö neljandas peatükis on kirjeldatud mobiilimängu arendust. Välja on toodud nii serveri, kui ka kliendi poolne lahendus. Mäng on loodud kasutades programmeerimiskeelt Kotlin ja serverilahendust Firebase. Autor on detailsemalt kirjeldanud mängu loomiseks kasutatud peamisi komponente ja nende kasutust, tuues näiteid valminud mängu programmikoodist. Valminud mängu kirjeldatakse ka lõppkasutaja vaateväljast.

# 2 Kotlin

Kotlin on avatud lähtekoodiga programmeerimiskeel, mis on suunatud Java platvormile. See on kokkuvõtlik, turvaline, pragmaatiline ja keskendunud kokkusobivusele. Seda saab kasutada peaaegu kõikjal, kus kasutatakse Javat – serveripoolsel arendusel, Androidi ja töölaua rakendustes ning mujal. Kotlinist genereeritakse tavaline Java baitkood ja see töötab ilma probleemideta juba kõikide olemasolevate Java teekide ja raamistikega. [18] Java on maailma üks populaarsemaid programmeerimiskeeli [19] ning Kotlin võimaldab arendajatel saavutada sama funktsionaalust vähemate koodiridade ja probleemideta.

Nagu Java, on ka Kotlin kompileeritav programmeerimiskeel. See tähendab, et enne programmikoodi jooksutamist on see vaja kompileerida. Kotlini lähtekoodi hoitakse failides, mille laiendiks on *.kt*. Kotlini kompilaator analüüsib lähtekoodi ja genereerib *.class* failid, täpselt nagu seda teeb Java kompilaator. Genereeritud *.class* failid pakitakse ja jooksutatakse kasutades standartseid meetodeid vastavalt rakenduse tüübile. Kotlini kompilaatoriga kompileeritud kood sõltub Kotlin *runtime* teegil, mis sisaldab endas definitsioone nii Kotlini enda teegi klassidest, kui ka laiendusi, mis Kotlin lisab standartsele Java API-le. Kotlin *runtime* teek tuleb pakkida koos rakendusega. (Joonis 1) [18]




Joonis 1. Kotlini ehitusprotsess

## 2.1 Eelised

### 2.1.1 Turvaline

Üks kõige tihedamini esinev probleem mitmetes programmeerimiskeeltes, kaasa arvatud Javas, on *null* viidete kasutamine, mille tulemusena tekivad programmi töös erindid, mis võivad viia rakenduse töö lõppemiseni. Kotlini tüübi süsteem on arendatud mõttega need erandid täielikult elimineerida. Seetõttu on võimalik eristada kahte tüüpi viiteid – neid, mis saavad hoida *null*'i (Joonis 2) ja neid, mis ei saa. (Joonis 3) [20]

```
var a: String = "kotlin"
a = null // kompileerimis viga
```

Joonis 2. Viide, mis ei saa hoida nulli

```
var b: String? = "kotlin"
b = null
```

Joonis 3. Viide, mis saab hoida nulli

Tänu Kotlini turvalisele tüübisüsteemile on võimalik viidetele teha ohutuid väljakutseid. Viidete poole pöördutakse vaid siis, kui antud viide pole *null*. Seda tehakse kasutades '?' operaatorit. (Joonis 4) Kui ollakse kindel, et viide pole kindlasti *null*, on võimalik seda kompilaatorile teada anda kasutades '!!' operaatorit, mis *null*'i leidmisel viskab programmi töös erindi. (Joonis 5)

```
var l = b?.length
```

Joonis 4. '?' operaatori kasutamine

```
var l = b!!.length
```

Joonis 5. '!!' operaatori kasutamine

Kotlini tüübisüsteem on mugav arendajatele, sest vead leitakse juba kompileerimise ajal. See tagab rakenduse parema töökindluse ja vähendab rakenduse töö käigus tekkivaid probleeme.

### 2.1.2 Kokkuvõtlik ja väljendav

Üldiselt loeb programmeerija programmikoodi rohkem, kui ta seda kirjutab. [21] Sellele põhimõttele on toetutud ka Kotlini väljaarendamisel. Mida lihtsam ja kokkuvõtlikum programmikood on, seda kiiremini saab selle funktsioonist aru. Lisaks sellele mängib rolli ka hea rakenduse arhitektuur ja väljendatavad nimetused. Programmeerimiskeel on

kokkuvõtlik, kui selle süntaks väljendab selgelt seda, mida on soovitud saavutada ning puudub vajadus kirjutada palju stereotüüpset ja sarnast programmikoodi. Kotlini arendusel on neid põhimõtteid silmas peetud ning üritatud programmikood võimalikult väljendusrikkaks teha. Paljude Java stereotüüpsete koodiridade elimineerimine on Kotlini üks positiivseid aspekte, mis vähendab tunduvalt vajaminevate koodiridade arvu. [18]

Nagu enamikel modernsetel programmeerimiskeeltel on ka Kotlinil rikas standartne teek, mis võimaldab asendada pikki ja korduvaid programmikoodi sektsioone teegi meetodite väljakutsetega. Kotlin toetab ka lambdasi, mis koondavad paljud tüüpilised programmikoodi osad Kotlini standartsesse teeki ja ainult programmi unikaalsed osad jäävad lähtekoodi alles. [18]

Joonis 6 näitab tüüpilise Java klassi implementeerimist Kotlinis. Kui Javas nõuab selle klassi implementeerimine ligi poolsada rida, siis Kotlinis on see võimalik ühe reaga, kasutades andme-klassi süntaksit. Kotlin genereerib tüüpilised vajaminevad meetodid automaatselt. [22]

```
data class Player(var id: Int,
                  var name: String,
                  var level: Int,
                  var skin: Int)
```

Joonis 6. Kotlini andmeklass

Kokkuvõtlikuma ja sisutihedama programmikoodi kirjutamine ja lugemine on vähem ajakulukas. See tõstab produktiivsust ja võimaldab eesmärgini jõuda kiiremini.

### 2.1.3 Ühilduv

Kotlin on täielikult ühilduv Javaga, võimalik on kasutada kõiki juba olemasolevaid Java teeke. See tähendab, et Kotlinist on võimalik lisaks muule välja kutsuda Java meetodeid, laiendada Java klasse, implementeerida liideseid ja kasutada Java annotatsioone. Erinevalt paljudest teistest JVM programmeerimiskeeltest, on võimalik välja kutsuda Kotlini koodi Javast, ilma, et selleks peaks lisa vaeva nägema. See pakub võimalust kasutada oma rakenduse arendamisel nii Javat kui ka Kotlinit segamini, mis lihtsustab Kotliniga programmeerimise alustamist. [18]

## 2.2 Puudused

Kuigi Kotlin on arendatud mõttega Java arendajate elu parandada, pole siiski suudetud kõiki murekohti likvideerida. Kotlini kasutamine toob endaga kaasa ka potentsiaalseid muresid. Suuremate androidi rakenduste puhul on vaja jälgida meetodite koguarvu, sest ühe rakenduse meetodite limiidiks on 65536. Seda ületades on vaja konfigureerida oma rakendust kasutama multidexi, mis toob endaga kaasa märkimisväärse kompileerimisaja pikenemise. [23] (Joonis 7) Kotlini teegi meetodite arv on umbes 6000, mis juba ainult üksinda moodustab üle 10 protsendi limiidist. Kahjuks ka ilma limiidi ületamiseta võtab Kotlini kompileerimine märgatavalt rohkem aega, kui Java. [24] Lisaks sellele suureneb ka rakenduse kasutatav andmemaht seadmes. Kotlini kasutamiseks vajamineva teegi suurus on ligikaudu 731 kilobaiti. [25]

Võrreldes Javaga on Kotlinil suuri puudujääke ka dokumentatsioonis. Kuigi sellega tegeletakse pidevalt, ei ole see veel võrreldaval tasemel Javaga, mis on ka arusaadav arvestades nende kahe programmeerimiskeele ajalugu.

```
trouble writing output:
Too many field references: 131000; max is 65536.
You may try using --multi-dex option.
```

Joonis 7. Androidi meetodite limiidi ületamine

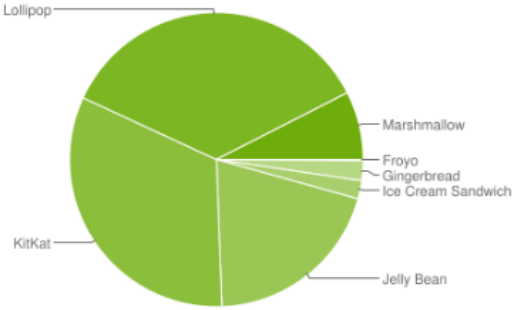
## 2.3 Kotlin ja Android

Tüüpiline Androidi rakendus erineb tunduvalt tüüpilistest suurettevõtete rakendustest. See on tavaliselt väiksem ja ei vaja integreerumist nii paljude juba eksisteerivate rakendustega. Väga tähtsal kohal on rakenduste töökindlus laial hulgal erinevatel seadmetel. Kotlini iseärasused koos spetsiaalse kompilaatori *plugin*-iga, mis toetavad Androidi platvormi, teevad Androidi arenduse palju produktiivsemaks ja nauditavamaks. Paljusid tüüpilisi programmeerimisülesandeid on võimalik lahendada vähemate koodiridade või isegi ilma koodi kirjutamata.

Kuna Java 8-ga on võimalik arendada vaid 7.5 protsendile Androidi kasutajatele [26] (Joonis 8), toob Kotlin Androidi arendusse palju uusi mugavaid tehnikaid: lambdad, *null*-turvalisus, laiendatavad funktsioonid, järelduslikud tüübid ja palju muud. Samal ajal ei too Kotlini kasutamine kaasa ühtegi ühilduvuse probleemi, sest Kotlin on täielikult ühilduv Java 6-ga. Kotliniga saab kasutada kõiki talle omaseid modernseid

programmeerimistehnikaid ning rakendust on võimalik jooksutada ka seadmetel, mis ei kasuta kõige uuemaid Androidi versioone. Jõudlust Kotlini kasutamine endaga kaasa ei too. Kotlini kompilaatori poolt genereeritud kood täidetakse sama tõhusalt, kui tavaline Java kood. Mõningatel juhtudel võib Kotlini kasutamine isegi kaasa tuua jõudluse kasvu. [18]

| Version | Codename | API | Distribution |
|---|---|---|---|
| 2.2 | Froyo | 8 | 0.1% |
| 2.3.3 - 2.3.7 | Gingerbread | 10 | 2.2% |
| 4.0.3 - 4.0.4 | Ice Cream Sandwich | 15 | 2.0% |
| 4.1.x | Jelly Bean | 16 | 7.2% |
| 4.2.x | | 17 | 10.0% |
| 4.3 | | 18 | 2.9% |
| 4.4 | KitKat | 19 | 32.5% |
| 5.0 | Lollipop | 21 | 16.2% |
| 5.1 | | 22 | 19.4% |
| 6.0 | Marshmallow | 23 | 7.5% |




Joonis 8. Androidi seadmete versioonid

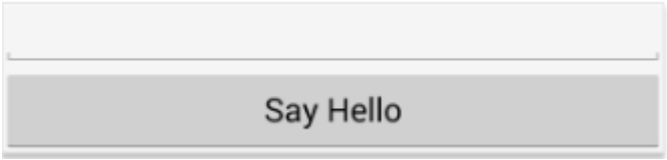
### 2.3.1 Anko

Anko on *DSL* Androidi jaoks. See on Kotlini autorite poolt arendatud võimekas teek, mis on kirjutatud Kotlinis ja mille peamiseks eesmärgiks on kasutajaliidese genereerimine. [22] Kui tüüpiliselt toimub Androidi androidi kasutajaliidese loomine kasutades XML'i, siis Anko abil on seda võimalik mugavalt teha programmikoodis. XML'i kasutamisel on omad puudused – see pole tüübikindel, *null*-kindel ja selle parsimine toimub otse seadmel, mis raiskab protsessori ressurssi ja akut. [27] Joonis 9 on näha Anko kasutamist kasutajaliidese loomisel ja Joonis 10 selle tulemust. Lisaks kasutajaliidese genereerimisele on teegis ka hulgaliselt laialtkasutatavaid meetodeid, mis lihtsustavad rakenduse programmikoodi kirjutamist. [28]

Anko suurimaks eeliseks on selle mugav kasutus. See võimaldab hoida kogu programmikoodi ühes kohas ning pakub võimalust koodi taaskasutamiseks. Lisaks , toetab see Kotlini üldist põhimõtet muuta kood loetavamaks ja sisutihedamaks.

```
verticalLayout {
    val name = editText()
    button("Say Hello") {
        onClick { toast("Hello, ${name.text}!") }
    }
}
```

Joonis 9. Anko kasutamine kasutajaliidese loomisel




Joonis 10. Anko genereeritud kasutajaliidese näide

# 3 Reaalaja serverilahendused

Reaalajas serverilahendused pakuvad mänguritele maagilise elamuse – kaks või enamat mängijat jagavad ühist kogemust läbi võrgu nagu nad eksisteeriksid samas virtuaalmaailmas. Seda võimaldavad erinevad reaalaja serverilahendused.

Esimestes reaalajas toimuvates mängudes kasutati P2P ühendusi. Põhiidee seisnes mängu tükeldamises üliväikesteks käikudeks ja käskudeks. Mängu toimimiseks oli vaja vaid need käivitada iga mängija masinas alustades identsest algseisust. Seda ühendust kasutatakse siiani mitmetes mängudes, kuid sellel esines palju puudujääke. Suurimateks miinusteks on raskus tagada ühtne algseis kõikidel masinatel ning vajadus oodata kõige aeglasema ühenduse järgi, mis halvendab mängu kasutaja kogemust kõikide mängijate jaoks. Nende probleemide lahendamiseks tuli muuta võrgu mudelit ning töödati välja klient-server mudel. Selle asemel, et kliendid suhtleksid otse omavahel, suhtlevad klient-server mudelis kõik kliendid ühe ja sama serveriga. Kadus vajadus, et mängud oleks igas kliendis identsed, sest ainus ’õige’ mänguseis on serveris, mida kõik kliendid kuvavad oma masinas. Puhtas klient-server mudelis ei jooksutata mängu koodi lokaalselt, vaid kõik sisendid saadetakse serverisse, mis uuendab selle põhjal andmeid serveris ning tagastab klientidele uue info mänguolukorrast. Selle mudeli kasutuselevõtuga tõusis reaalaja mängude kvaliteet märgatavalt, sest mäng ei sõltunud enam kõige aeglasemast kliendist, vaid ühenduse kiirusest vastava kliendi ja serveri vahel. [29]

Järgnevalt analüüsib autor mitmeid serveripoolseid teenuseid ning platvorme, mis lihtsustavad reaalajas mängu loomist.

## 3.1 Firebase

Firebase on erinevate pilveteenuste ja serveripoolsete lahenduste pakkuja. Ettevõte pakub mitmeid tooteid ning teenuseid tarkvaraarendajatele loomaks veebi- ning mobiilirakendusi. [30]

Põhiliseks teenuseks on reaalajas andmebaas pilves, mis pakub arendajatele mugavat API-t, mis võimaldab andmeid talletada ning sünkroniseerida üle mitme seadme. Teenuse geniaalsus seisneb andmete hoidmise ja jagamise liitmises üheks süsteemiks.

Firebase ei liiguta lihtsalt informatsiooni ühelt kliendilt teisele, vaid lisab kõikide klientide andmed tsentraalsesse andmehoidlasse, mis asub pilves ning mida kõik kliendid jälgivad. Kui üks klient muudab hoidlas andmeid, sünkroniseeritakse see kõikides klientides. Arendaja vastutab ainult selle eest, et kliendid jälgiksid õiget andmehoidlat ning kogu andmete transport toimub automaatselt. [31] Firebase pakub kliendipoolseid teeke, mis võimaldavad integratsiooni rakendustega, mis on ehitatud platvormidele nagu Android, iOS, JavaScript, Java, Objective-C ja Node.js. Andmebaas on kättesaadav ka REST API kaudu. [32] REST API kasutab SSE protokolli, mis on API loomaks HTTP ühendusi, et vastu võtta reaalajas teateid serverilt. Arendajad, kes kasutavad reaalajas andmebaasi, saavad oma andmete turvalisuse tagada kasutades ettevõtte serveri poolt jõustatud turva reegleid. [33]

Firebase hoiab andmeid suure JSON dokumendina, nagu seda teevad ka enamus NoSQL andmebaasi süsteeme. Andmeid hoitakse kui suuri objekte, mis suudavad hoida ’võti-väärtus’ paare, kus väärtus saab olla sõna, number või teine objekt. Andmete mugavaks haldamiseks on reaalajas uuendav veebirakendus. [34]

Lisaks sellele pakub Firebase ka veebimajutust ning autentimise teenust. [30]

Positiivseteks omadusteks keskkonna puhul on kindlasti mugavus. Reaalajas rakenduse kirjutamiseks ei pea arendaja kirjutama ise üldse serveripoolset koodi. See kiirendab rakenduse loomise protsessi märgatavalt. Firebase’i poolt pakutav veebirakendus on väga mugav oma andmete haldamiseks. Samuti ei pea muretsema andmete hoidmise pärast, sellega tegeleb suurepäraselt Firebase poolt pakutav pilvelahendus.

Negatiivne antud platvormi puhul on see, et puudub võimalus muuta serveripoolset struktuuri. Kuigi JSON andmeformaat on laialt levinud ning mugav, puudub arendajal siiski täielik kontroll serveripoolse andmetalletuse üle ning pole võimalik vastavalt oma vajadustele süsteemi muuta.

## 3.2 Meteor

Meteor on tasuta ja avatud lähtekoodiga veebiraamistik, mis on kirjutatud Node.js-is. See võimaldab ülikiiret rakenduste prototüüpimist ja pakub platvormide (veeb, Android, iOS) vahelist sünkroniseerimist. Kogu süsteem integreerub MongoDB andmebaasiga. Meteor kasutab DDP protokolli ja avalda–telli mustrit automaatseks andmete

uuendamiseks klientide rakendustes ilma, et arendaja peaks kirjutama sünkroniseerimiseks spetsiaalset programmikoodi. Kliendi poolelt on Meteor sõltuv populaarsest teegist jQuery-st ja on kasutatav ükskõik millise JavaScripti kasutajaliidese teegiga. [35]

Meteori eeliseks on võimalus arendada mitmele platvormile korraga, mis teeb erinevatele platvormidele arendamise kiiremaks ja odavamaks. Samas tuleb ise programmeerida ka serveri pool, mis ei sea arendajale serveripoolseid piiranguid, kuid aeglustab rakenduse loomise protsessi. Puuduseks on ka Meteori raske integreerimisprotsess mitte-Javascripti kasutavate rakendustega nagu Android'i või iOS'i põlisrakendustega.

## 3.3 Socket.IO

Socket.IO on JavaScripti teek ehitamaks reaalaja rakendusi. See võimaldab reaalajas kahepoolset kommunikeerimist kliendi ja serveri vahel. See koosneb kahest komponendist: kliendipoolne teek mis jookseb rakenduse poolel ja serveripoolne teek node.js jaoks. Mõlemal komponendil on peaaegu identne API. Socket.IO on sündmustele põhinev. [36]

Socket.IO kasutab põhiliselt WebSocket protokolli, mis vajadusel taandub teistele protokollidele. Selletõttu on arendajal vaja teadmisi vaid Socket.IO'st, kuid seab kohustuse kasutada seda nii serveri, kui kliendi poolel. Androidi jaoks on võimalik kasutada Socket.IO Java kliendi teeki. [37]

Socket.IO eelisteks on hea dokumentatsioon ja aktiivne kogukond. Samuti on seda ka lihtne kasutada nii serveri kui kliendi poolel paralleelselt.

## 3.4 Feathers.js

Feathers.js on minimalistlik JavaScripti raamistik. See on paindlik reaalaja raamistik, mida saab kasutada nii brauseri kui ka serveri poolel. Feathers on ehitatud kasutades ES6, mis lubab arendajatel kasutada kõiki moodsaid JavaScripti võimalusi ja kirjutada kokkuvõtvat ja elegantset koodi. Feathers on väga hästi integreeruv erinevate kliendipoolsete raamistikega nagu näiteks React, Angular ja React Native. [38]

Dokumentatsioonis on väga hästi kirjeldatud nende integreerimist, mis teeb kogu protsessi väga mugavaks.

Feathers on teenustele orienteeritud. Teenused on raamistiku põhiidee. Need pakuvad reaalajas kohest vastust erinevatele meetoditele: otsi, võta, loo, uuenda, paranda ja kustuta. Kõikidele nendele tegevustele serveri poolel saab reaalajas koheselt kliendi poolel reageerida. Kuna Feathers töötab kasutades teenuseid, pakub see lisaks reaalaja API'le ka REST API, mis töötab automaatselt üle HTTP/HTTPS ja üle *websocket*'ite. Lisaks sellele on Feathersil adapterid üle viieteistkümnele erinevale serveri lahendusele, mis annab võimaluse kasutada muude hulgas selliseid populaarseid andmebaase nagu Postgres, MySQL, SQLite, MongoDB. [38]

Feathers'i suurimaks eeliseks võibki lugeda selle mugavat kasutust koos juba olemasolevate andmebaasidega. Lisaks sellele on Feathers'il väga hea dokumentatsioon, mis teeb arendamise palju mugavamaks. Negatiivse poolena on vaja palju arendust teha serveri poolel. Lisaks üldisele serveripoolsele arendusele on vaja see ära ühendada Feathersiga.

# 4 Implementatsioon

Selles peatükis analüüsib autor loodud rakenduse peamisi komponente ja nende arendust. Kirjeldatud on nii kasutatud tehnoloogia teoreetilist poolt, kui ka näiteid reaalsest rakenduse programmikoodist.

## 4.1 Server

Eelnevalt on punktis 3 kirjeldatud mitut erinevat reaalaja serverilahendust. Igal lahendusel on omad positiivsed ja negatiivsed omadused olenevalt arendaja soovidest ja rakenduse spetsiifikast. Autor valis mängu loomiseks serverilahenduse Firebase, eelistades seda selle mugavuse pärast.

### 4.1.1 Struktuur

Kõiki Firebase'i andmebaasi andmeid hoitakse JSON objektina, puuduvad tabelid ja tabeliread. Võimalikud andmetüübid on String, Boolean, Long, Double, Map<String, Object> ja List<Object>. [39] Mängu käigus kasutatud andmeobjekti atribuute on kirjeldatud Tabel 1 ja reaalset näidist JSON objektist kirjeldab Joonis 11.

Tabel 1. Ühe mängu andmeobjekti atribuudid

| **Atribuudi nimi** | **Tüüp** | **Kirjeldus** | **Näide** |
|---|---|---|---|
| Player | Objekt | Mängijat kirjeldav objekt | {id: 33, x: 100.53, y: 200.53, lives: 2} |
| Player.name | String | Mängija nimi | „Mart Maasikas“ |
| Player.id | Long | Mängija unikaalne id | 33 |
| Player.x | Double | Mängija x koordinaadi asukoht, teisendatuna üldisele suurusele | 532.123 |
| Player.y | Double | Mängija y koordinaadi asukoht, teisendatuna üldisele suurusele | 532.123 |
| Player.lives | Long | Mängija elude arv | 3 |

| **Atribuudi nimi** | **Tüüp** | **Kirjeldus** | **Näide** |
|---|---|---|---|
| Player.powerup | Long | Mängija aktiivne lisa | -1 |
| Playercount | Long | Mängijate arv konkreetses mängus | 2 |
| Gamerunning | Boolean | Tõeväärtus, mis näitab, kas mäng on käimas | True |
| Projectiles | Objekt | Objekt, mis hoiab kõiki mängus kasutatud laskude objekte | {} |
| Projectiles.Projectile | Objekt | Ühte lasku kirjeldav objekt | {playerid: 33, startx: 202.20, starty: 202.20, endx: 213.20, endy: 230.24} |
| Projectile.playerid | Long | Lasu autori id | 33 |
| Projectile.startx | Double | Lasu alguse x koordinaadi asukoht, teisendatuna üldisele suurusele | 202.20 |
| Projectile.starty | Double | Lasu alguse y koordinaadi asukoht, teisendatuna üldisele suurusele | 250.20 |
| Projectile.endx | Double | Lasu lõpu x koordinaadi asukoht, teisendatune üldisele suurusele | 400.34 |
| Projectile.endy | Double | Lasu lõpu y koordinaadi asukoht, teisendatune üldisele suurusele | 332.35 |

| **Atribuudi nimi** | **Tüüp** | **Kirjeldus** | **Näide** |
|---|---|---|---|
| Powerups | Objekt | Objekt, mis hoiab kõiki mängu lisade objekte | {powerup1,powerup2} |
| Powerups.Powerup | Objekt | Ühte lisa kirjeldav objekt | {id: 2, x: 320.34, y: 543.34, used: false} |
| Powerup.id | Long | Lisa tüübi identifikaator | 2 |
| Powerup.x | Double | Lisa x koordinaadi asukoht, teisendatuna üldisele suurusele | 220.43 |
| Powerup.y | Double | Lisa y koordinaadi asukoht, teisendatuna üldisele suurusele | 220.43 |
| Powerup.used | Boolean | Tõeväärtus, mis näitab kas lisa on kasutatud või mitte | True |

```
{
	"player1" : {
		"name" : „Mart“,
		"id" : 23,
		"x" : 528.23111,
		"y" : 343.24334,
		"lives" : 3,
		"powerup" : -1
	},
	"player2" : {
		"name" : „Mari“,
		"id" : 25,
		"x" : 128.54231,
		"y" : 223.24334,
		"lives" : 2,
		"powerup" : 2
	},
	"gamerunning" : true,
	"playercount" : 2,
	"projectiles" : {
	    "-KHsfFywJRd33rIc-sEG" : {
		  "endx" : 248.30602,
		  "endy" : 129.28572,
		  "playerid" : 24,
		  "startx" : 500.25247,
		  "starty" : 322.0
		},
		"-KHsfGlYOeVUm7Qgyy0W" : {
		  "endx" : 270.78845,
		  "endy" : 117.80167,
		  "playerid" : 25,
		  "startx" : 527.0,
		  "starty" : 322.0
		}
	},
	"powerups" : {
		"-KHsfHqTzRG97bqLOK3o" : {
		  "id" : 2,
		  "x" : 229.52322,
		  "y" : 24.43234,
		  "used" : false
		}
	}
}
```


Joonis 11. Näidis mänguobjekt JSON formaadis

### 4.1.2 Kasutus

Selleks, et andmed reaalajas kliendini jõuaks, on vaja kuulata serveri muudatusi. Kui mingi väärtus serveris muutub, saadetakse see kohe kliendile, kus seda siis vastavalt kasutatakse. Firebase'l on selle jaoks mugav API. Joonis 12 on näide, kuidas kuulata serveri poolseid muutuseid Firebase's.

```
val myFirebaseRef =
       Firebase("https://kotlingame.firebaseio.com/games")
              .child(gameIdentifier)
myFirebaseRef!!.addValueEventListener(
                    object : ValueEventListener {
     override fun onDataChange(snapshot: DataSnapshot) {

          var p1x : Double? = snapshot
                                .child("player1")
                                .child("x")
                                .value as? Double;
          var p1y : Double? = snapshot
                                .child("player1")
                                .child("y")
                                .value as? Double;
          updatePlayerPosition(p1x?.toInt(), p1y?.toInt())
          var p2x : Double? = snapshot
                                .child("player2")
                                .child("x")
                                .value as? Double;
          var p2y : Double? = snapshot
                                .child("player2")
                                .child("y")
                                .value as? Double;
          updatePlayer2Position(p2x?.toInt(), p2y?.toInt())
          var p1lives : Long? = snapshot
                                .child("player1")
                                .child("lives")
                                .value as? Long
          var p2lives : Long? = snapshot
                                .child("player2")
                                .child("lives")
                                .value as? Long
          updatePlayerLives(p1lives, p2lives)
 })
```

Joonis 12. Firebase serveri kuulamine

## 4.2 Rakendus

Rakendus on kirjutatud täielikult programmeerimiskeeles Kotlin.

Rakendus on sõltuv järgnevatest teekidest:

- Kotlini standartne teek versioon 1.0.1-2
- Firebase Androidi kliendi teek versioon 2.5.2
- Anko versioon 0.8.3
- Android Support teek versioon 23.2.1

Vajalikud on järgmised load:

- PERMISSION_NFC – luba rakendusel kasutada NFC'd
- PERMISSION_INTERNET – luba rakendusel suhelda internetiga
- PERMISSION_VIBRATE – luba rakendusel seadet vibreerida

Järgnevalt kirjeldab autor lähemalt nelja põhilist komponenti, mille realiseerimise tulemusena valmis mobiilimäng.

### 4.2.1 SurfaceView

Mängu erinevate elementide kuvamiseks ekraanile kasutatakse klassi nimega *SurfaceView*. *SurfaceView* pakub võimalust ekraanile joonistada kõiksuguseid elemente, kasutamata seadme põhilõime. See tähendab, et rakendus ei pea ootama, kuni süsteemi vaated on valmis joonistama, vaid joonistamine toimub teises lõimes. [40] Selleks, et ekraanile joonistada teiselt lõimelt, tuleb esialgu lõuend lukustada kasutades *lockCanvas* meetodit. Kui lõuendile joonistamine on tehtud, tuleb välja kutsuda meetod *unlockCanvasAndPost.* Selle tagajärjel joonistatakse ekraanile valminud lõuend. Iga kord, kui lõuend lukustatakse, jääb sinna alles eelmine seisund. Selle tõttu tuleb korralikult animeerimise jaoks iga kord uuesti joonistada kogu lõuend. [41] (Joonis 13)

```
private fun draw() {
    while (!holder.surface.isValid) {
        try {
            Thread.sleep(10)
        } catch (e: InterruptedException) {
            e.printStackTrace()
        }
    }
    val canvas = holder.lockCanvas()
    drawCanvas(canvas)
    holder.unlockCanvasAndPost(canvas)
}

private fun drawCanvas(canvas: Canvas) {
    background.draw(canvas)
    player.draw(canvas)
    player2.draw(canvas)
    obstacle.draw(canvas)
    projectiles.forEach { if(it.visible) it.draw(canvas) }
    powerups.forEach { if(!it.used!!) it.draw(canvas) }
}
```

Joonis 13. Lõuendile joonistamine kasutades SurfaceView'd

### 4.2.2 Graafika, animeerimine

Graafika ja animatsiooni implementeerimiseks kasutatakse *sprite*'sid. *Sprite* on üksik graafiline pilt, mis moodustab ühe osa kogu ekraani pildist. See on populaarne viis loomaks komplekssseid ja mahukaid ekraanipilte, sest võimaldab manipuleerida igat *sprite*'i eraldi. [42] Androidi platvormil kasutatakse *sprite* kuvamiseks klassi Bitmap. Selleks, et erinevate suurustega ekraanidel oleks pilt proportsionaalselt sama suur ning ei võtaks ülemäära palju ressurssi, tuleb seda esmalt töödelda. Selleks kasutatakse klassi BitmapDecoder. (Joonis 14) Kui bitmappe ei kasutata õigesti, võivad need kiiresti kasutada ära kogu seadme olemasoleva mälumahu, mis viib rakenduse sulgumiseni. [43].

Mängud koosnevad tavaliselt suurest hulgast *sprite*'dest. Nende ükshaaval laadimine ja protsessimine võtaks meeletult mälukasutust. Selle probleemi lahendamiseks kasutatakse *spritesheet*'e, mis kujutab endas pilti, millel on mitu *sprite*'i. Palju kiirem on laadida pilt ühe korra ja näidata sellest ekraanile vaid väikest osa, kui laadida ja protsessida iga pilt eraldi. [42]

Joonis 15 on autori loodud programmikood, mida kasutatakse rakenduses *spritesheet*'i animeerimiseks. Mängu käigus kutsutakse konstantselt välja meetodit *moveToNextFrame*, mis valib *spritesheet*'ilt järgmise pildi asukoha, mis siis joonistatakse ekraanile kasutades *draw* meetodit.

```
bitmap = Util.getDownScaledBitmapAlpha8(game, R.drawable.heart)

fun getDownScaledBitmapAlpha8(context: Context, id: Int): Bitmap {
    val bitmapOptions = BitmapFactory.Options()
    bitmapOptions.inPreferredConfig = Bitmap.Config.ALPHA_8
    bitmapOptions.inScaled = true
    bitmapOptions.inDensity = DEFAULT_DENSITY
    bitmapOptions.inTargetDensity =
         Math.min((getScaleFactor(context) *
         DEFAULT_DENSITY).toInt(), DEFAULT_DENSITY)
    val b = BitmapFactory.decodeResource(
         context.resources, id, bitmapOptions)
    b.density = context.resources.displayMetrics.densityDpi
    return b
}
```

Joonis 14. Bitmapi loomiseks loodud meetod ja selle väljakutse

```
abstract var bitmap: Bitmap

abstract var width: Int
abstract var height: Int
abstract var x: Float
abstract var y: Float

// Spritesheet'i ridade ja veergude arv
abstract var colNr: Int
abstract var rowNr: Int

var col: Int = 0
var row: Int = 0
var src = Rect() // The source frame of the bitmap
var dst = Rect() // The destination area

open fun draw(canvas: Canvas) {
    src.set(col * width, row * height,
                (col + 1) * width, (row + 1) * height)
    dst.set(x.toInt(), y.toInt(),
                (x + width).toInt(), (y + width).toInt())
    canvas.drawBitmap(bitmap, src, dst, null)
}

open fun moveToNextFrame() {
    if(++col == colNr) {
        col = 0;
        if(++row == rowNr)
            row = 0;
    }
}
```

Joonis 15. Sprite animeerimiseks loodud programmikood

### 4.2.3 Aktseleromeeter ehk kiirendusmõõtur

Valminud mängus on karakteri liigutamiseks on vaja kallutada oma seadet. Selle funkstionaalsuse realiseerimiseks kasutas autor aktseleromeetrit ehk kiirendusmõõturit. Aktseleromeeter on seade, mis mõõdab kiirendust kolmel teljel [44], kuid antud mängu kontekstis oli vaja vaid kahte. Selleks, et saada seadmelt infot kiirenduste kohta eri telgedel, tuleb implementeerida *SensorEventListener* liidest. Iga kord, kui kiirendus mõnel teljel muutub, kutsutakse välja *onSensorChanged* meetod ja sellelt loetud andmete põhjal uuendatakse mängu olukorda. (Joonis 16)

```
override fun onSensorChanged(event: SensorEvent?) {
    if(calibrating) {
        myInitialSensorX = event!!.values[1]
        myInitialSensorY = event!!.values[0]
    } else if (isGameRunning != null && isGameRunning!!) {
        game!!.sensorEvent(event!!.values[1] - myInitialSensorX,
                    event.values[0] - myInitialSensorY)
    }
}
```

Joonis 16. Aktseleromeetri väärtuste lugemine

### 4.2.4 Android Beam

Android Beam on NFC tehnoloogia, mis lubab rakendustel saata infomatsiooni teistele seadmetele. Selleks, et NFC töötaks, peavad seadmed olema üksteisele vähemalt 4 sentimeetri kaugusel. Üks rakendus loob sõnumi ja määrab kindlaks, millised rakendused on võimelised seda sõnumit kätte saama. Kui kindlaksmääratud rakendus on aktiivne ja seadmed on üksteisele piisavalt lähedal, saadab Android Beam sõnumi teisele seadmele, mis seejärel protsessib kättesaadud sõnumit. [45] Android Beam töötab Androidi seadmetel alates versioonist 4, mis on kasutusel üle 95 protsendile Androidi kasutajatele. [26] Lisaks on vaja deklareerida AndroidManifestis luba kasutada NFC'd.

Loodud rakenduses kasutab autor Android Beami, et jagada kahe seadme vahel unikaalset mängu identifikaatorit. Seda indentifikaatorit on vaja, et mõlemad seadmed oleksid ühenduses ühe ja sama mänguobjektiga serveris. Identifikaatori genereerimiseks kirjutatud klass asub Joonis 17. Funktsioon *getRandomString* illustreerib ka Kotlinile omast *single-expression* funktsiooni.

```
class RandomStringGenerator {
    var random = SecureRandom()

    fun getRandomString(): String =
            BigInteger(130, random).toString(32)
}
```

Joonis 17. Mängu identifikaatori genereerimine

Android Beami kasutav rakendus peab implementeerima liideseid *NfcAdapter.CreateNdefMessageCallback ja OnNdefPushCompleteCallback.* Selleks, et

saata sõnum ühelt seadmelt teisele on vaja luua *NdefMessage*, mis sisaldab *NdefRecord*'it, kasutades *createNdefMessage* meetodit. [46] (Joonis 18)

```
override fun createNdefMessage(event: NfcEvent): NdefMessage {
    var gameIdentifier = generateGameId()
    val msg = NdefMessage(
            arrayOf(createMimeRecord(
                    "application/com.suislepp.andres.kotlingame",

                      gameIdentifier.toByteArray()),
                      NdefRecord.createApplicationRecord(
                            "com.suislepp.andres.kotlingame"))
    )
    return msg
}

fun createMimeRecord(mimeType: String, payload: ByteArray): NdefRecord {
    val mimeBytes = mimeType.toByteArray(Charset.forName("US-ASCII"))
    val mimeRecord = NdefRecord(
            NdefRecord.TNF_MIME_MEDIA, mimeBytes, ByteArray(0), payload)
    return mimeRecord
}
```

Joonis 18. Android Beami sõnumi loomine

Sõnumi kättesaamiseks on vaja AndroidManifestis määrata activity'd, mis tegelevad Androidi Beami sõnumi vastu võtmisega. (Joonis 19) Kui rakendusele tuleb *intent*, mille teguviis on NDEF_DISCOVERED, delegeeritakse see vastavale *activity*'le, kus sellega siis tegeletakse. (Joonis 20)

```
<intent-filter>
    <action android:name="android.nfc.action.NDEF_DISCOVERED"/>
    <category android:name="android.intent.category.DEFAULT"/>
    <data android:mimeType=
        "application/com.suislepp.andres.kotlingame"/>
</intent-filter>
```

Joonis 19. Android Beami sõnumi vastuvõtu delegeerimine

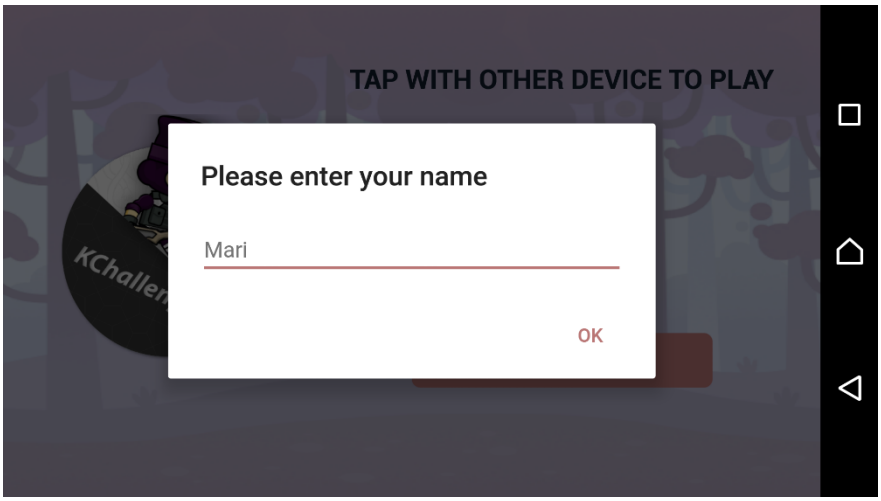
```
override fun onResume() {
    super.onResume()
    if (NfcAdapter.ACTION_NDEF_DISCOVERED == intent.action) {
        processIntent(intent)
    }
}

internal fun processIntent(intent: Intent) {
    val rawMsgs = intent.getParcelableArrayExtra(
            NfcAdapter.EXTRA_NDEF_MESSAGES)
    val msg = rawMsgs[0] as NdefMessage
    processMessage(msg)
}
```

Joonis 20. Android Beami sõnumi vastuvõtmine

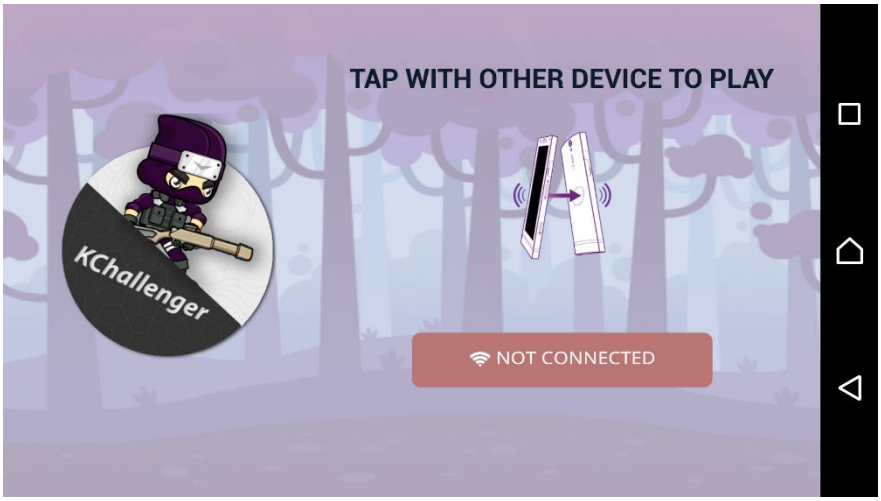
## 4.3 Valminud mäng

Autor arendas mobiilimängu Androidi platvormile kasutades programmeerimiskeelt Kotlin. Serverilahendusena kasutas autor Firebase'i. Kogu mängu loomiseks kasutatud graafika on kas autori omalooming või avalik ja tasuta. Mängu mängimiseks peab seadmel olema NFC tugi, aktselerомeeter ja internetiühendus.

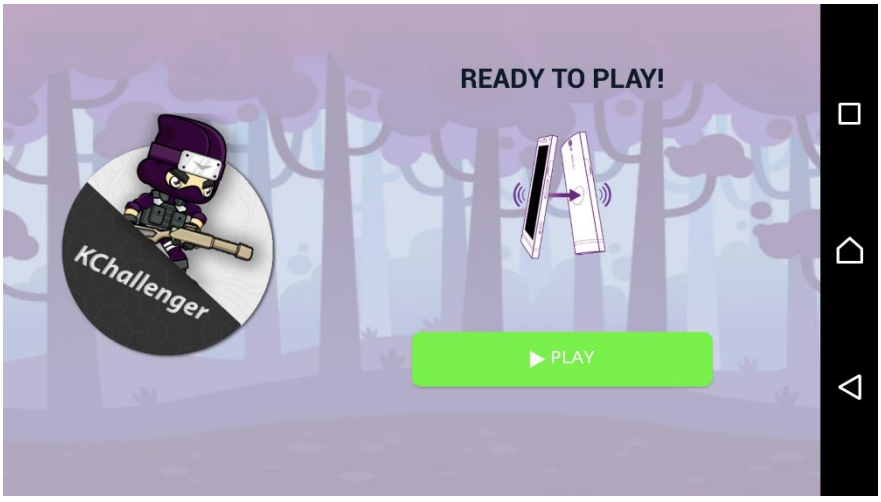
Mobiilimängu eesmärk on võita oma vastast lahingus. Esmasel mängu avamisel palutakse mängijal sisestada oma alias. (Joonis 21) Mängu alustamiseks on vaja ühendada ennast oma vastasega. (Joonis 22) Selleks peavad kaks mängijat viima oma telefonid lähestikku, mille tagajärjel ühinevad nad ühe ja sama serveriga. (Joonis 23)Kui mõlemad mängijad on serveriga ühendanud (Joonis 24), algab mäng reaalajas üksteise vastu. (Joonis 25) Selleks, et vastast võita, tuleb teda tabada laskudega. Oma karakterit saab liigutada kallutades oma telefoni. Laskmiseks on vaja puudutada oma seadme ekraani, mis paneb paika suuna, kuhu lask suundub. Kui ei õnnestu vastase lasu eest ära põigelda, kaotab mängija ühe elu. Mängija, kes kaotab esimesena kolm elu, kaotab. Mängule lisavad põnevust erinevad takistused ja boonused, mida on võimalik kokkupuutel enda karakteriga saada.



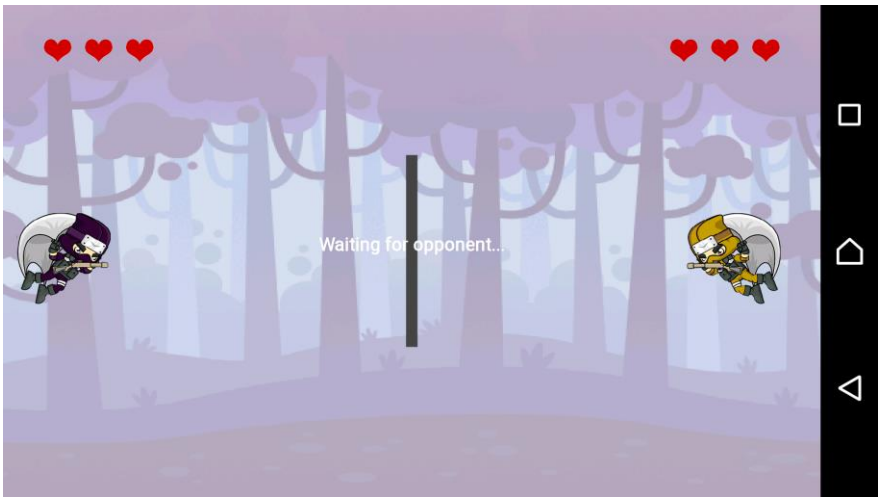

Joonis 21. Ekraanipilt aliase loomisest



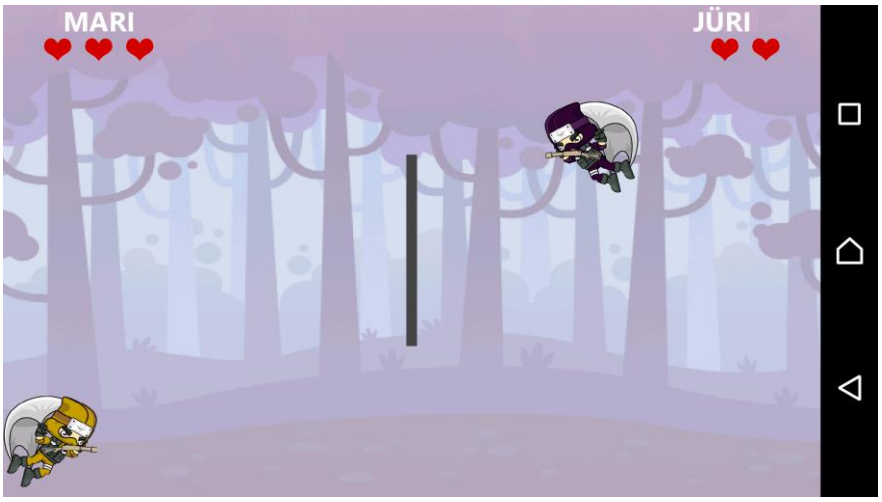

Joonis 22. Ekraanipilt menüüst, kui ühendus teise seadmega puudub




Joonis 23. Ekraanipilt menüüst, kui ühendus teise seadmega on olemas




Joonis 24. Ekraanipilt vastasmängija ootamisest




Joonis 25. Ekraanipilt lahinguolukorrast

## 4.4 Hinnang Kotlinile

Alustades Kotliniga programmeerimist on suureks abiks eelnev kogemus programmeerimiskeelega Java. Neil on sarnane süntaks ja neid on võimalik kasutada paralleelselt. Kuna Androidi platvormile arendamiseks on siiani kasutusel Java versioon 6, pakub Kotlin võimalust kasutada mitmeid populaarseid ja modernseid tehnikaid nagu lambdad, laiendatavad funktsioonid, järelduslikud tüübid ja palju muud. See muudab arendusprotsessi märgatavalt nauditavamaks. Kotlinis kirjutatud kood väldib stereotüüpseid ja korduvaid koodiridu, mille tagajärjeks on kergemini loetav ja arusaadav programmikood. Kotlini suurimaks puuduseks peetavat ülemäära pikka kompileerimiseks kuluvat aega autor oma töö käigus ei täheldanud.

Autor hindab Kotlini kasutamise kogemust üksnes positiivseks. Kotliniga oli mugav mobiilimängu arendada ning valminud programmikood oli loetav ja kokkuvõtlik. Hoolimata Kotlini suhteliselt noorele vanusele, leidus erinevatele probleemidele internetis piisavalt vasteid, et leida lahendus. Kotlini kasutamine ei seadnud ühtegi piirangut rakenduse arendamiseks. Selle tõttu arvab autor, et Kotlin on tänasel päeval parim programmeerimiskeel arendamaks põlisrakendust Androidi platvormile.

# 5 Kokkuvõte

Käesoleva bakalaureusetöö eesmärkideks oli:

- analüüsida Kotlini võimalusi Androidi platvormil;
- uurida reaalaja serverilahendusi;
- arendada kahe mängijaga reaalajas toimiv mobiilimäng, kasutades selleks programmeerimiskeelt Kotlin.

Autor analüüsis Kotlini programmeerimiskeelt, selle ehitusprotsessi ja kasutust, tõi välja selle positiivsed ja negatiivsed omadused võrreldes neid programmeerimiskeelega Java. Kotlini suurimaks eeliseks võib lugeda tema turvalisust ning selle kasutamisega tõuseb rakenduse töökindlus. Autor leidis ka, et Kotlini programmikood on kokkuvõtlik ja hästi loetav ning see ühildub probleemideta juba olemasolevate Java teekidega. Kotlini suurimate probleemidena tõi autor välja kompileerimiseks kuluva aja märgatava suurenemise ning kasina dokumentatsiooni.

Autor uuris lisaks reaalaja mängude loomise metoodikat ja erinevaid olemasolevaid võimalusi. Autor puutus kokku serverilahendustega Firebase, Meteor, Socket.IO ja Feathers.js. Arvestades eelnevate lahenduste eeliseid ja puuduseid, valis autor rakenduse loomiseks serverilahenduse Firebase, eelistades seda tema mugava ja kiire kasutuse poolest.

Kasutades programmeerimiskeelt Kotlin valmis töö raames mobiilimäng Androidi platvormile. Mängu objektide ekraanile kuvamiseks kasutati SurfaceView klassi ja animeerimiseks *spritesheet*'e. Karakteri liigutamise funktsionaalsuse lõi autor kiirendusmõõturi abiga. Kahe erineva mängija ühendamiseks rakendati NFC'l põhinevat tehnoloogiat Android Beam. Autor hindas Kotlini kasutust üksnes positiivseks kogemuseks.

Eelnimetatud teemasid analüüsides ja loodud mobiilimängu põhjal saab väita, et bakalaureusetöö eesmärk on täidetud. Töö võimalikeks edasiarendusteks võiks olla arendatud mängule võimalus, et mäng toimuks rohkem kui kahe mängija vahel. Lisaks võiks integreerida mängu näiteks *Google Play Services* keskkonnaga ja luua kõikide kasutajate vahel üldine edetabel.

# Kasutatud kirjandus


[1] Google, „App Manifest,“ [Võrgumaterjal]. https://developer.android.com/guide/topics/manifest/manifest-intro.html. [Kasutatud 11 Mai 2016].

[2] Wikipedia, „Rakendusliides,“ [Võrgumaterjal]. https://et.wikipedia.org/wiki/Rakendusliides. [Kasutatud 02 Mai 2016].

[3] Android Picks, [Võrgumaterjal]. http://androidpicks.com/explain-apk/. [Kasutatud 02 Mai 2016].

[4] Wikipedia, „Publish–subscribe pattern,“ 2016. [Võrgumaterjal]. https://en.wikipedia.org/wiki/Publish%E2%80%93subscribe_pattern. [Kasutatud 28 Aprill 2016].

[5] S. Stubailo, „DDP Specification,“ 2014. [Võrgumaterjal]. https://github.com/meteor/meteor/blob/master/packages/ddp/DDP.md. [Kasutatud 28 Aprill 2016].

[6] A. v. Deursen, P. Klint ja J. Visser, „Domain-Specific Languages: An Annotated Bibliography,“ [Võrgumaterjal]. http://www.st.ewi.tudelft.nl/~arie/papers/dslbib.pdf. [Kasutatud 01 Mai 2016].

[7] Wikipedia, „Client,“ [Võrgumaterjal]. https://en.wikipedia.org/wiki/Client_(computing). [Kasutatud 02 Mai 2016].

[8] Wikipedia, „NFC,“ [Võrgumaterjal]. https://et.wikipedia.org/wiki/NFC. [Kasutatud 02 Mai 2016].

[9] B. Wen, „6 things you should know about Node.js,“ *JavaWorld,* 2013. [E-ajakiri] http://www.javaworld.com/article/2079190/scripting-jvm-languages/6-things-you-should-know-about-node-js.html [Kasutatud 05 Mai 2016]

[10] Wikipedia, „Null,“ [Võrgumaterjal]. https://en.wikipedia.org/wiki/Null. [Kasutatud 02 Mai 2016].

[11] Wikipedia, „P2P-võrgustik,“ [Võrgumaterjal]. https://et.wikipedia.org/wiki/P2P-v%C3%B5rgustik. [Kasutatud 02 Mai 2016].

[12] Wikipedia, „Cloud Storage,“ [Võrgumaterjal]. https://en.wikipedia.org/wiki/Cloud_storage. [Kasutatud 02 Mai 2016].

[13] Wikipedia, „Plug-in (computing),“ [Võrgumaterjal]. https://en.wikipedia.org/wiki/Plug-in_(computing). [Kasutatud 02 Mai 2016].

[14] Wikipedia, „REST,“ [Võrgumaterjal]. https://et.wikipedia.org/wiki/REST. [Kasutatud 02 Mai 2016].

[15] I. Hickson, „Server-Sent Events,“ *W3C,* 2015. [E-ajakiri] https://www.w3.org/TR/eventsource/ [Kasutatud 09 Aprill 2016]

[16] Wikipedia, „Library_(computing),“ [Võrgumaterjal]. https://en.wikipedia.org/wiki/Library_(computing). [Kasutatud 11 Mai 2016].

[17] A. Melnikov ja I. Fette, „The WebSocket Protocol,“ [Võrgumaterjal].

https://tools.ietf.org/html/rfc6455. [Kasutatud 11 Mai 2016].

[18] D. Jemerov ja S. Isakova, Kotlin In Action, Manning Publications, 2016.

[19] M. Weinberger, „The 9 most popular computer languages, according to the Facebook for programmers,“ *Business Insider,* 2015. [E-ajakiri] http://www.businessinsider.com/github-most-popular-coding-languages-2015-8 [Kasutatud 27 Aprill 2016]

[20] JetBrains, „Null-Safety,“ [Võrgumaterjal]. https://kotlinlang.org/docs/reference/null-safety.html. [Kasutatud 14 Aprill 2016].

[21] R. Chen, „Code is read much more often than it is written, so plan accordingly,“ 2007. [Võrgumaterjal]. https://blogs.msdn.microsoft.com/oldnewthing/20070406-00/?p=27343/. [Kasutatud 10 04 2016].

[22] A. Leiva, Kotlin for Android Developers, Lean Publishing, 2016.

[23] Google, „Multidex,“ [Võrgumaterjal]. https://developer.android.com/studio/build/multidex.html. [Kasutatud 29 Aprill 2016].

[24] M. Gouline, „Kotlin, the Swift of Android,“ 2016. [Võrgumaterjal]. https://blog.gouline.net/kotlin-the-swift-of-android-d664f8997e7f#.o2b5x9d7t. [Kasutatud 29 Aprill 2016].

[25] P. Torchinsky ja S. Isakova, „Getting started with Android and Kotlin,“ [Võrgumaterjal]. https://kotlinlang.org/docs/tutorials/kotlin-android.html. [Kasutatud 29 Aprill 2016].

[26] Google, „Platform Versions,“ [Võrgumaterjal]. http://developer.android.com/about/dashboards/index.html. [Kasutatud 08 Mai 2016].

[27] JetBrains, „Anko,“ [Võrgumaterjal]. https://github.com/Kotlin/anko. [Kasutatud 28 Aprill 2016].

[28] JetBrains, „Announcing Anko for Android,“ *JetBrains Blog,* 2015. [E-ajakiri] http://blog.jetbrains.com/kotlin/2015/04/announcing-anko-for-android/ [Kasutatud 22 Aprill 2016]

[29] G. Fiedler, „What every programmer needs to know about game networking,“ *GAFFER ON GAMES,* 2015. [E-ajakiri] http://gafferongames.com/networking-for-game-programmers/what-every-programmer-needs-to-know-about-game-networking/ [Kasutatud 30 Aprill 2016]

[30] Google, „Firebase Official Website,“ Firebase, [Võrgumaterjal]. https://www.firebase.com/. [Kasutatud 12 Aprill 2016].

[31] C. Metz, „‘Firebase’ Does for Apps What Dropbox Did for Docs,“ *Wired,* 2012. [E-ajakiri] http://www.wired.com/2012/04/firebase/ [Kasutatud 25 Aprill 2016]

[32] Google, „Firebase Docs,“ [Võrgumaterjal]. https://www.firebase.com/docs/. [Kasutatud 02 Aprill 2016].

[33] B. Darrow, „Firebase secures its real-time back-end service,“ *Gigaom,* 2012. [E-ajakiri] https://gigaom.com/2012/12/18/firebase-secures-its-real-time-back-end-service/ [Kasutatud 26 Aprill 2016]

[34] R. Khanna, „Structuring your Firebase Data correctly for a Complex App,“ *Airpair.* [E-ajakiri] https://www.airpair.com/firebase/posts/structuring-your-firebase-data [Kasutatud 25 Aprill 2016]

[35] Meteor, „Meteor Official Website,“ [Võrgumaterjal]. https://www.meteor.com/.

[Kasutatud 29 Aprill 2016].

[36] R. Guillermo, „Socket.IO,“ [Võrgumaterjal]. http://socket.io/docs/. [Kasutatud 02 Mai 2016].

[37] Socket.IO, „Socket.IO-client Java,“ [Võrgumaterjal]. https://github.com/socketio/socket.io-client-java. [Kasutatud 02 Mai 2016].

[38] E. Kryski, „Introducing Feathers 2.0: A minimalist real-time framework for tomorrow’s apps,“ [Võrgumaterjal]. https://blog.feathersjs.com/introducing-feathers-2-0-aae8ae8e7920#.77htto7kd. [Kasutatud 08 Mai 2016].

[39] Google, „Understanding Data,“ [Võrgumaterjal]. https://www.firebase.com/docs/android/guide/understanding-data.html. [Kasutatud 02 Mai 2016].

[40] Google, „SurfaceView,“ [Võrgumaterjal]. https://developer.android.com/reference/android/view/SurfaceView.html. [Kasutatud 13 Mai 2016].

[41] Google, „2D Graphics,“ [Võrgumaterjal]. https://developer.android.com/guide/topics/graphics/2d-graphics.html. [Kasutatud 13 Mai 2016].

[42] S. Lambert, „An Introduction to Spritesheet Animation,“ *Tutsplus,* 2013. [E-ajakiri] http://gamedevelopment.tutsplus.com/tutorials/an-introduction-to-spritesheet-animation--gamedev-13099 [Kasutatud 16 Mai 2016]

[43] Google, „Displaying Bitmaps Efficiently,“ [Võrgumaterjal]. https://developer.android.com/training/displaying-bitmaps/manage-memory.html. [Kasutatud 15 Mai 2016].

[44] Xamarin, „Accelerometer,“ [Võrgumaterjal]. https://developer.xamarin.com/recipes/android/os_device_resources/accelerometer/. [Kasutatud 11 Mai 2016].

[45] Xamarin, „Android Beam,“ [Võrgumaterjal]. https://developer.xamarin.com/guides/android/platform_features/android_beam/. [Kasutatud 03 Mai 2016].

[46] Google, „NFC Basics,“ [Võrgumaterjal]. http://developer.android.com/guide/topics/connectivity/nfc/nfc.html#p2p. [Kasutatud 02 Mai 2016].

[47] M. Lehenbauer, „Developers, meet Firebase!,“ *Firebase Blog,* 2012. [E-ajakiri] https://www.firebase.com/blog/2012-04-12-developers-meet-firebase.html [Kasutatud 12 Aprill 2016]